OULUN AMMATTIKORKEAKOULU

Juha Jämsä

KIIHTYVYYSANTURIIN PERUSTUVA AJOTOTTUMUSTEN TARKKAILUJÄRJESTELMÄ ANDROID-KÄYTTÖJÄRJESTELMÄLLE

KIIHTYVYYSANTURIIN PERUSTUVA AJOTOTTUMUSTEN TARKKAILUJÄRJESTELMÄ ANDROID-KÄYTTÖJÄRJESTELMÄLLE

Juha Jämsä
Opinnäytetyö
Kevät 2018
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Juha Jämsä
Opinnäytetyön nimi: Kiihtyvyysanturiin perustuva ajotottumusten tarkkailujärjestelmä Android-käyttöjärjestelmälle
Työn ohjaaja: Pekka Alaluukas
Työn valmistumislukukausi ja -vuosi: Kevät 2018
Sivumäärä: 42

Opinnäytteen tavoitteena oli luoda Android-käyttöjärjestelmälle sovellus joka analysoi kiihtyvyysanturin dataa ja tämän perusteella ilmoittaa käyttäjälle liiallisista kiihtyvyyden muutoksista ajoneuvoa kiihdytettäessä tai jarrutettaessa. Tilaajana toimi Oulun ammattikorkeakoulu.

Sovelluksen avulla käyttäjä voi tarkastella kiihtyvyysanturin dataa erilaisin graafein reaaliajassa ja tallentaa dataa tiedostoon myöhempää analysointia varten.

Toiminnallisuus on toteutettu Java-ohjelmointikielellä Android Studio -ohjelmointiympäristössä.

Sovellusta olisi jatkokehitettävä edelleen huomattavasti, jotta siitä olisi minkäänlaista käytännön hyötyä. Tällä hetkellä sen pääasiallinen tarkoitus on vain suuntaa antava, projektin alkuperäisten vaatimusten mukaisesti.

Asiasanat: ajotapa, Android, mobiililaitteet

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author(s): Juha Jämsä
Title of thesis: Acceleration sensor based driving habit monitoring system for Android-operating system
Supervisor(s): Pekka Alaluukas
Term and year when the thesis was submitted: Spring 2018
Number of pages: 42

The purpose of this thesis was to create an application for Android operating system that analyzes accelerometer data and notifies the user of excessive acceleration changes when the vehicle is accelerating or braking. Oulu University of Applied Sciences acts as a subscriber.

The application allows the user to monitor accelerometer data with variety of graphs in real time and also includes the ability to save the data in a file for later analysis.

The functionality of the application was created in Java-programming language, in Android Studio development environment.

The application should be developed further to gain any kind of practical use out of it. As its current state its main purpose is mainly indicative, according to the projects original requirements.

Keywords: driving, Android, mobile devices

SISÄLLYS

SANASTO

.apk	Androidin pakettitiedosto, joka sisältää asetukset ja Java-käännöksen koodista.
Activity	Aktiviteetti. Android-järjestelmässä yleensä koko ruudun peittävä, tietyt toiminnallisuudet sisältävä, interaktiivinen näkymä.
Android	Linux-pohjainen, mobiililaitteissa yleisesti käytetty käyttöjärjestelmä
Android Studio	Android-sovellusten ohjelmointiympäristö
Avoin lähdekoodi	Vapaasti saatavilla, kaikkien tarkasteltavissa oleva koodi
Google	Yhdysvaltalainen yhtiö joka vastaa Android-järjestelmän kehityksestä, myös hakukone
GPS	Global Positioning System, Yhdysvaltain puolustusvoimien kehittämä satelliittipaikannusjärjestelmä
Java	Ohjelmointikieli, yleisesti käytössä Android-sovellusten tuottamisessa
Käyttöjärjestelmä	Keskeinen tietokoneen ohjelmisto, joka hallinnoi koneen resursseja, mahdollistaa muiden ohjelmien toiminnan
Linux	Linus Torvaldsin Unixin modulaarisuuden mukaan kehittämä avoimen lähdekoodin käyttöjärjestelmä.
Kernel	Käyttöjärjestelmän perustoiminnallisuuden sisältävä ydin.
Mobiili	Mukana kulkeva
Mobiiliapplikaatio	Sovellus, joka on suunniteltu mobiilipäätelaitteisiin
Mobiilipäätelaite	Tabletti, matkapuhelin, älypuhelin

Ohjelmointiympäristö	Ohjelmistopaketti, joka sisältää työkaluja koodin tuottamiseen, kääntämiseen ja testaamiseen.
Open source	Avoin lähdekoodi
OpenSUSE	Linux-jakelu
Partitio	Tallennusmuistin osio. Näkyy käyttöjärjestelmälle erillisenä fyysisenä tallennusvälineenä.
Samsung	Korealainen elektroniikkavalmistaja
Tabletti	Käytännössä suurinäyttöinen matkapuhelin, jonka pääasiallinen käyttöasento on vaakataso.
Unix	Bell Labsin 1970-luvulla kehittämä, modulaarinen patentoitu käyttöjärjestelmä.
Windows	Microsoftin hallinnoima, maksullinen, suljetun lähdekoodin käyttöjärjestelmä

1 JOHDANTO

Työn tavoitteena oli toteuttaa Android-mobiilisovellus, joka kiihtyvyysanturin perusteella tarkkailisi ajoneuvon kuljettajan ajotapaa liian kovien kiihdytysten ja jarrutusten kannalta. Näistä tiedoista olisi mahdollista antaa jonkinlaista arviota myös polttoaineen kulutuksesta.

Päättötyön aihetta pohtiessani entinen koulutusohjelmavastaava Riitta Rontu osoitti minulle tekstitiedoston, johon oli listattuna useita erilaisia aiheita, joiden toteutusmahdollisuuksia joku tuntematon henkilö oli joskus pohtinut. Tuosta listasta tämä aihe valikoitui verrattain helposti, sillä olin tuolloin hiljattain perehtynyt juuri Android-sovelluksiin ja jonkin verran Android-laitteissa oleviin sensoreihin. Syvempi pureutuminen Android-laitteisiin ja niiden ominaisuuksiin sekä jonkinlaisen konkreettisen testin laatiminen oli ajatustasolla kiehtonut jo hetken aikaa. Myös mielenkiinto halpaan sensoriin perustuvan mittausjärjestelmän mahdollisuuksista innosti ryhtymään tähän projektiin. Omasta ajotavasta olin myös hieman huolestunut, joten ajattelin, että jos tällainen sovellus olisi mahdollista saada toimimaan, siitä saattaisi olla oikeastikin hyötyä. Kiihtyvyysanturilla tehtävistä mittauksista sekä projektiin liittyvästä fysiikan teoriasta minulla ei kuitenkaan ollut juuri käytännön kokemusta, joten projekti eteni ikään kuin havaintoja tehdessä ja niistä oppiessa.

2 TYÖKALUT

2.1 OpenSUSE

OpenSUSE on avoimen lähdekoodin Linux-käyttöjärjestelmä. OpenSUSE pohjautuu saksalaiseen SUSE Linux -käyttöjärjestelmään, mikä siirtyi Novell-yhtiön omistukseen vuonna 2003. Novell ilmoitti siirtävänsä käyttöjärjestelmän kehitystä avoimempaan, yhteisökehitteiseen suuntaan ja uudelleen nimesi SUSE-käyttöjärjestelmän openSUSE:ksi. Attachmate osti Novellin vuonna 2011 ja erotti SUSEn Novellista omaksi yksikökseen. Mirco Focus International plc osti Attachmaten vuona 2014. (1.)

OpenSUSE:n kehittäminen tapahtuu avoimesti yhteisöpohjaisena. Tämä tarkoittaa, että kuka vain voi osallistua järjestelmän kehitystyöhön ja kaikki käyttöjärjestelmään liittyvä koodi on vapaasti kaikkien luettavissa. OpenSUSE:n tarkoituksena on olla helposti käyttöön otettava ja helppokäyttöinen, karsimatta kuitenkaan edistyneempiä toimintoja. (2.)

2.2 Android Studio

Android Studio on IntelliJ IDEA -kehitysympäristöön pohjautuva, virallinen integroitu kehitysympäristö (IDE) Android-sovellusten kehittämistä varten. Se sisältää kaikki tarvittavat komponentit mm. koodieditori, kääntämiseen liittyvät työkalut, Android-emulaattori, testaustyökalut, käyttöliittymäeditori ja niin edelleen. (3.)

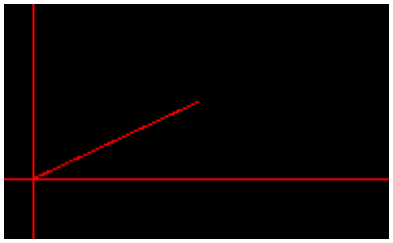
2.3 VeusZ

VeusZ on avoimen lähdekoodin työkalu datan visualisoimiseen. Ohjelmaan voi tuoda dataa erilaisista formaateista ja datasta saa luotua halutunlaisen kuvaajan. Dataan voi sovittaa myös funktioita. Ohjelma on erinomainen lineaarisen datan analysoinnissa ja erilaisten suodattimien kehittelyssä. Ohjelmalla luodut

graafit voidaan myös ”viedä” erilaisiin kuvaformaatteihin. Ohjelman toiminnallisuutta voidaan laajentaa edelleen Python-lisäkkeillä. Ohjelmasta löytyy myös skriptauskäyttöliittymä, jonka avulla Python-kieltä osaava voi käsitellä dataa mielin määrin. Tavallisen graafin luominen ei kuitenkaan vaadi minkäänlaista Python-osaamista. Ohjelma on saatavilla Unix/Linux-, Windows- sekä MacOS-käyttöjärjestelmille. (4.)

3 SOVELLUKSEN RAKENNE

Aloitin työn suunnittelun perehtymällä Android-laitteiden kiihtyvyysanturin dokumentaatioon ja sen toimintaan. Minulle oli täysin epäselvää, kuinka työ tulisi suorittaa ja minkälaista dataa olisi ylipäätänsä mahdollista saada tehtävämäärityksessä mainitusta kiihtyvyysanturista. Kun olin jonkin verran perehtynyt Android-järjestelmän dokumentaatioon, tein aluksi pienen kokeellisen sovelluksen, jossa ei näkynyt ruudulla muuta kuin laidasta laitaan menevät punainen vaakaviiva, punainen pystyviiva sekä näiden viivojen risteämiskohtaan näytön keskipisteestä lähtevä viiva (kuva 1). Horisontaaliviivan piirtokoordinaatit otin kiihtyvyysanturin Y-akselin datasta ja vertikaaliviivalle X-akselilta, jotta saisin jonkinlaisen konkreettisen käsityksen anturin toiminnasta.



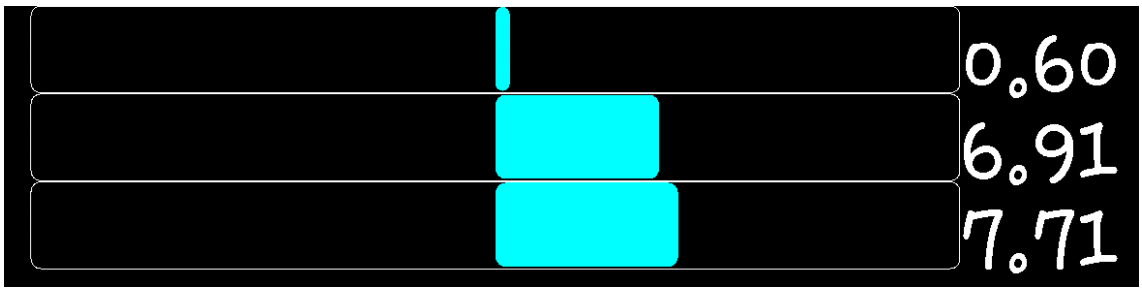
KUVA 1. Kohinan ilmaisin

3.1 Suodatin

Välittömästi ohjelman ajettuani havaitsin, että minulla on todellinen ongelma kiihtyvyysanturin kohinan suhteen, jos aion saada sovelluksesta hyödyllistä mittausdataa. Punaisten viivojen muodostama risti väpätti ympäri ruutua varsin vallattomasti. Oli selvää, että järkevän datan saamiseksi tarvitsen jonkinlaisen suodattimen, jolla häiriötä saisi pienemmäksi. Ajattelin, että toistaiseksi riittänee ihan perussuodatin, joka yksinkertaisesti laskee keskiarvon muutamasta kiihtyvyysanturin antamasta näytteestä. Fiksumman suodattimen voisin askarrella sitten, kun ohjelma on siinä vaiheessa, että datan analysointi on ajankohtaista.

3.2 Indikaattorit

Seuraavaksi piirsin näytölle edistymispalkkien kaltaiset indikaattorit (kuva 2) joille voin antaa arvot erikseen kaikilta kolmelta akselilta, numeeristen arvojen kera. Indikaattoreiden maksimin sain suoraan kiihtyvyysanturilta.



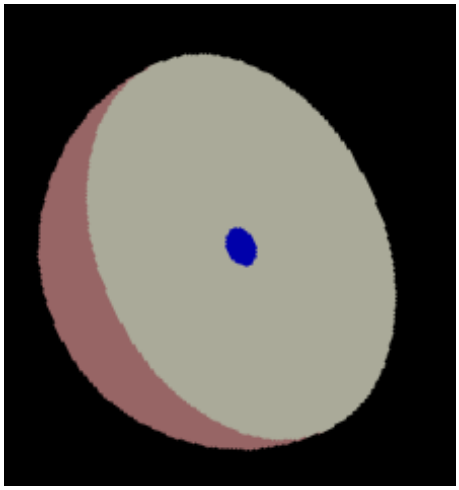
KUVA 2. Edistymispalkki indikaattorit

Kyseiset indikaattorit selkeyttivät kovasti anturilta saatavia lukemia mutta myös lisäsivät hämmennystä. Pelkästään niitä tuijottamalla oli vaikea ymmärtää, mihin suuntaan kiihtyvyysvoima mobiililaitteen näkökulmasta sijoittuu.

3.3 Painovoima

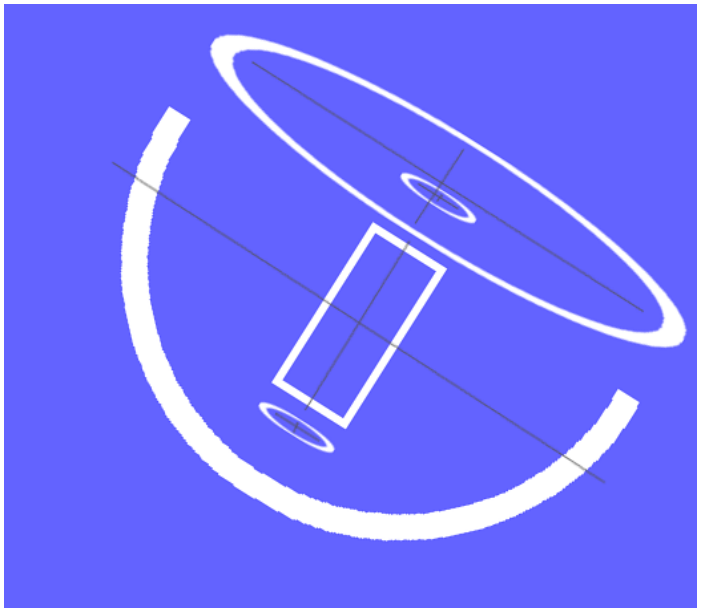
Kiihtyvyysanturi näyttää levossa vähintään siihen kohdistuvan, planeetan massasta johtuvan vetovoiman. Tämä on ilmeisesti asia, joka saattaa tulle joillekin yllätyksenä, sillä olen muutamaan otteeseen joutunut selittämään kuinka kiihtyvyysanturi ei ole viallinen vaan siihen kohdistuu painovoima.

Selkeyttääkseni voiman suuntaa kolmiulotteisessa avaruudessa tarvitsisin jonkin näköisen kuvaajan joka havainnollistaisi painovoiman suuntaa suhteessa mobiililaitteen asentoon. Päädyin piirtämään pseudokolmiulotteisen puolipallon (kuva 3).



KUVA 3. Puolipalloindikaattori

Puolipallon geometria muuttuu reaaliajassa siten että paksumpi puolisko osoittaa aina siihen suuntaan mihin kohdistuu suurin kiihtyvyys. Tämä elementti ei ole oikeasti kolmiulotteinen vaan se on yhdistelmä erilaisia kaksiulotteisia piirroksia (kuva 4) joiden asentoa muutetaan sopivasti, jotta saadaan aikaan illuusio kolmiulotteisesta elementistä.



KUVA 4. Puolipalloindikaattorin räjäytyskuva

3.4 Datan käsittely

Kun olin saanut rakennettua riittävästi havainnollistavia elementtejä ruudulle, alkoi kiihtyvyysanturin datan tulkitseminen olla huomattavasti selkeämpää. Tarvitsin kuitenkin jonkinnäköistä testidataa, jotta voisin analysoida dataa todellisesta kulkuneuvon käyttö tilanteesta. Androidin dokumentaatiosta tutkin, kuinka voisin tallettaa dataa tiedostoon.

Tiedoston kirjoitusta pohtiessa tulee valita, kirjoitetaanko sisäiseen vai ulkoiseen muistiin. Varhaisissa Androidin versioissa sisäinen muisti tarkoitti laitteen sisäänrakennettua muistia, johon voidaan pysyvästi tallentaa tietoa, ja ulkoinen vuorostaan tarkoitti muistikorttia tai muuta vastaavaa USB-muistia. Sisäisen ja ulkoisen muistin eroja ovat mm. sellaiset seikat, että sisäistä muistia ei voi selata esimerkiksi tietokoneella. Sovelluksilla on pääsy sisäisessä muistissa ainoastaan sovelluksen omiin tiedostoihin. Sisäinen muisti ei sen vuoksi sovellu datan tallentamiseen, kun tarkoitus oli analysoida sitä tietokoneella. Olisin voinut valita tallennusmuistiksi myös fyysisen muistikortin mutta en luottanut muistikorttini kirjoitusnopeuteen. Nykyään Android-laitteissa molemmat muistit ovat saatavilla vaikka fyysistä muistikorttia ei olisikaan. Tiedot tallennetaan laitteen sisäiseen muistiin molemmissa tapauksissa mutta ulkoinen muisti saattaa olla osa sisäistä muistia joka on allokoitu erilliseksi levyosioksi. Tämä tarkoittaa että ulkoinen muisti saattaa olla irrotettuna tiedostojärjestelmästä, esimerkiksi tietokoneeseen kytkemistä varten. Tämän vuoksi ennen ulkoiseen muistiin kirjoittamista tulee tarkistaa, että ulkoinen muisti on saatavilla ja siihen on mahdollista kirjoittaa. Dokumentaation mukaan tämä tapahtuu seuraavalla metodilla:

```
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

Jotta sovelluksella olisi lupa kirjoittaa ulkoiseen muistiin, täytyy sovelluksen AndroidManifest.xml -tiedostoon kirjoittaa pyyntö:

```
<manifest ...>
    <uses-permission android:name="
          android.permission.WRITE_EXTERNAL_STORAGE
     " />
    ...
</manifest>
```

Jos pyyntöä ei lisätä, järjestelmä estää ohjelman pääsyn ulkoiseen muistiin. (5.) Tiedostoon kirjoitus tapahtuu tiivistetysti näin:

```
File path = new File(
                     Environment.getExternalStoragePublicDirectory(
                     Environment.DIRECTORY_DOWNLOADS)
                     , "GasSniffer");
File file = new File(path,"data_"+Long.toString(elapsedRealTime)+".csv");


       try {
           filewriter = new FileWriter(file,true);
           filewriter.write( Long.toString(elapsedRealtime()-elapsedRealTime)+";"
              +Float.toString(x)+";"
              +Float.toString(y)+";"
              +Float.toString(z)+";"+"\n");
           filewriter.close();
       } catch (IOException e) {
           e.printStackTrace();
       }
```

Tiedosto tallennetaan ”Lataukset/GasSniffer/” -hakemistoon. Tallennettavan datan muotoa en ajatellut sen kummemmin. Määrittelin tiedostolle nimen: data_XXXXXXXXXX.txt jossa X-kirjaimet korvataan mobiililaitteen edellisestä käynnistyksestä kuluneella millisekuntimäärällä. Pitemmän päälle tämä on todella huono tapa, sillä jos käy niin epätodennäköinen tilanne, että laite uudelleenkäynnistetään ja aloitetaan tallentaminen juuri samalla millisekunnilla kuin

edellisellä uudelleenkäynnistyskerralla, niin uusi data kirjoitetaan vanhaan tiedostoon jatkoksi. Esimerkiksi päivämäärä ja kellon aika olisi hyvä lisä kyseiselle numerosarjalle. Alun perin tiedoston formaatti oli yksinkertaisuudessaan seuraava:

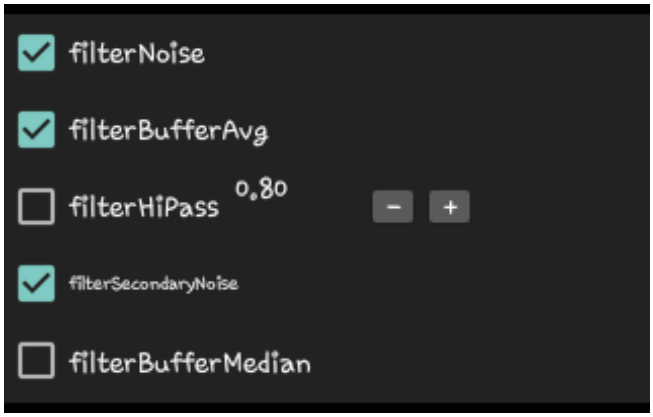
```
3|-0.060544662|1.661576|9.336189
5|-0.060544662|1.661576|9.336189
7|-0.060544662|1.661576|9.336189
9|-0.060544662|1.661576|9.336189
10|-0.060544662|1.661576|9.336189
12|-0.060544662|1.661576|9.336189
14|-0.060544662|1.661576|9.336189
15|-0.060544662|1.661576|9.336189
...
```

Data oli jaettu sarakkeisiin, jotka erottelin merkillä ”|” (pipe). Ensimmäisessä sarakkeessa on aika nauhoituksen aloituksesta, toisessa X-akselin lukema, kolmannessa Y-akseli ja neljännessä Z-akseli. Viimeisenä merkkinä joka rivillä on näkymätön ”\n” joka merkitsee rivin vaihtoa Linux-ympäristössä. Windowsissa vastaava rivinvaihto on ”\r\n” ja Applen käyttöjärjestelmissä ”\r”. Datantallennusformaattini muuttui myöhemmissä vaiheissa siten että sarakkeet erotellaan puolipisteellä ”;” ja tiedostopääte on .csv. Tämän tyyppisenä data on hivenen kätevämpi tuoda erilaisille ohjelmille analysoitavaksi. Myös datasarakkeita olen lisännyt ja poistanut testaamisen edetessä tarpeen mukaan.

3.5 Datan analysointi

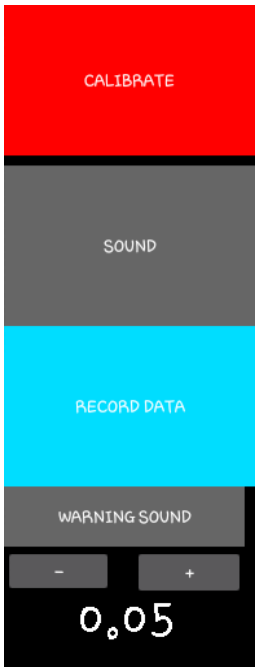
Dataa analysoidessa en ollut vielä tyytyväinen suodattimeni toimintaan ja kokeellisessa mielessä tein muutaman suodattimen lisää. Osa suodattimista kuitenkin osoittautui verrattain hyödyttömiksi sillä ne eivät selkeyttäneet dataa juuri laisinkaan vaan lisäsivät reaaliaikaista viivettä suotta. Jätin ohjelmaan viisi suodatinta testitarkoituksessa. Kolme näistä kuitenkin käytännössä osoittautui datan kannalta käyttökelpoiseksi ja ne ovat perusasetuksiltaan päällä ohjelmaa

käynnistettäessä. Suodattimen saa päälle ja pois asettamalla tai poistamalla suodattimen kohdalla rastin valintaruutuun (kuva 5):



KUVA 5. Suodatinvalinnat

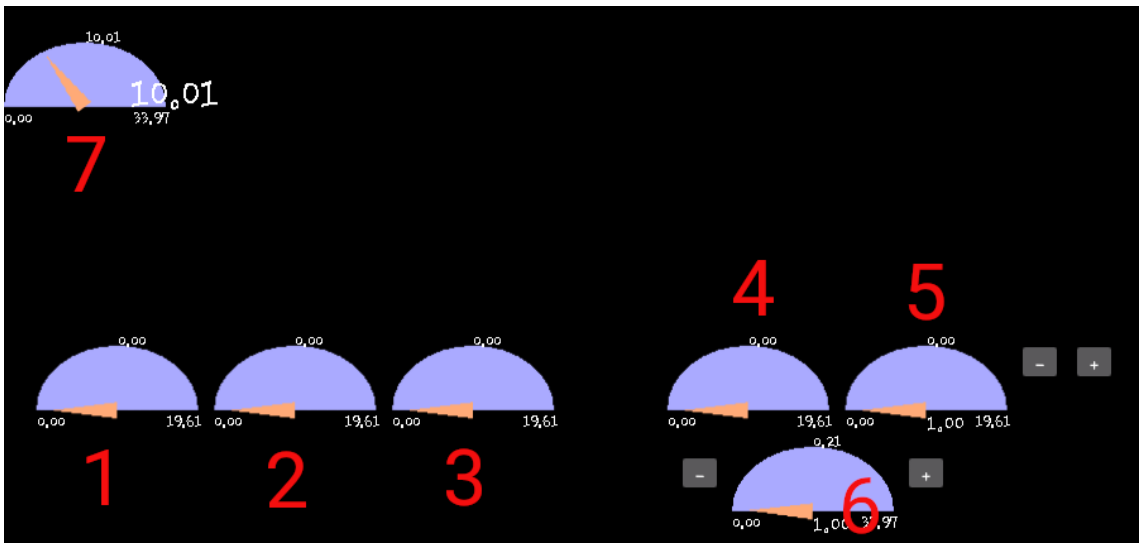
Tein ohjelmaan kalibrointitoiminnon, jonka toimintatarkoituksena on tasata jokainen akseli nollakohtaan reaaliaikaisen tarkastelun helpottamiseksi. Ilman kalibrointia ainoa mahdollisuus kiihtyvyys datan muutoksien tarkkailuun on vähentää painovoimakomponentti akseleiden magnitudista eli mittapisteen etäisyydestä akseleiden muodostamassa kolmiulotteisessa koordinaatistossa. Kalibrointi tulee suorittaa siten, että mobiililaite asetetaan paikkaan, jossa se ei pääse liikkumaan, ja pidetään samassa paikassa niin kauan, kuin tarkoitus on mittauksia tehdä. Jos laite liikkuu, on kalibrointi suoritettava uudestaan. Kalibrointi käynnistetään punaisesta painikkeesta (kuva 6), joka muuttuu vihreäksi kalibroinnin ollessa valmis. Ohjelma ei tiedä, milloin laite tahattomasti pääsee liikkumaan, eikä täten muuta vihreää painiketta takaisin punaiseksi, jos vahinko tapahtuu eikä kalibrointi enää päde.



KUVA 6. Hallinta painikkeita

3.6 Reaaliaikainen testaus

Reaaliaikaista testausta varten tein sarjan mittareita, joista osa näyttää anturin datan suoraan anturin minimin ja maksimin välillä, toinen mittari laskee akseleiden magnitudin ja osa on erinäisiä testimittareita, joiden näyttämää dataa olen vaihdellut miten milloinkin (kuva 7).



KUVA 7. Mittareita

Mittarit 1–5 näyttävät nollaa, jos kalibrointia ei ole suoritettu. Mittarit 1–3 ovat järjestyksessä kiihtyvyysanturin akseleille X, Y ja Z. Kalibroinnin jälkeen mittareiden 1–3 viisarit osoittavat suoraan ylöspäin lepoasennossa. Kiihtyvyyden muuttuessa viisarit liikkuvat puolelta toiselle riippuen siitä, muuttuuko kiihtyvyys positiiviseksi vai negatiiviseksi verrattuna akselin kalibroinnin myötä saatuun nollakohtaan. Akseli, jolla havaitaan suuri kiihtyvyyden muutos, vaihtaa taustavärin punaiseksi. Mittari 4 on akseleiden magnitudi, nollakohta huomioon ottaen. Mittari 5 näyttää tarkoituksella hieman vahvistettua magnitudin kaltaista lukemaa havainnollistamisen vuoksi. Mittari 7 näyttää jatkuvasti kaikkien akseleiden magnitudin. Mittari 6 vuorostaan magnitudin josta on painovoimakomponentti vähennetty. Kalibroinnin myötä mittari 6 on hyödytön eikä sen arvoa enää päivitetä.

Mittareiden arvoja oli kuitenkin todella vaikea seurata autolla ajaessa, joten tein luokan, jolla voidaan generoida äänimerkki. Sovelluksesta kuuluu kahdentyyppistä ääntä. Toinen ääni seuraa jatkuvasti akseleiden magnitudia ja vaihtaa taajuutta magnitudin mukaan. Toinen taas on niin sanottu varoitusääni, joka antaa äänimerkin, kun magnitudi ylittää jonkin tietyn raja-arvon, joka on säädettävissä mittareiden 5 ja 6 läheisyydessä näkyvistä +- ja –-painikkeista. Varoitusääni on sidottu suoraan ennen kalibrointia mittarin 6 ja kalibroinnin jälkeen mittarin 5 lukemiin.

Perusasetukseltaan äänet eivät ole päällä vaan ne aktivoidaan kuvassa näkyvistä painikkeista (kuva 6). Painikkeen väri muuttuu vihreäksi indikoiden kyseisen äänisignaalin olevan päällä. Vaikka tämän voi normaalisti kuulla korvin, on silti hyvä visualisoida tilanne sillä mobiililaite saattaa olla mykistettynä. Alimpana kuvassa on lukema sekuntimäärästä jota voi säätää kuvan +- ja –-painikkeilla. Tämä aika määrä otetaan huomioon, kun tutkitaan mittareiden painikkeilla asetettua tasoa, jolloin varoitusääni kuuluu. Aikasäädön tarkoituksena on neutralisoida piikkejä G-voimissa esimerkiksi kuoppaan ajettaessa. Jos kiihtyvyyspiikki on lyhempi kuin asetettu aika, ei äänimerkkiä kuulu.

3.7 Käyttöliittymä

Projektia aloittaessani minulla ei ollut muita testilaitteita kuin virtuaalinen Android-emulaattori ja fyysinen tabletti, joten käyttöliittymä oli suunniteltu pelkästään tätä yhtä tablettia ajatellen. Myöhemmässä vaiheessa sain käsiini myös Android-matkapuhelimen ja havaitsin että kiihtyvyys anturin X ja Y akselit olivat eri järjestyksessä sovelluksen orientaation suhteen. Matkapuhelinta on yleensä tarkoitus käyttää pystyasennossa ja tablettia vaaka-asennossa. Tämän takia matkapuhelinta varten piti tehdä erillinen painike, jolla akselit saadaan käännettyä kohdalleen (kuva 8).

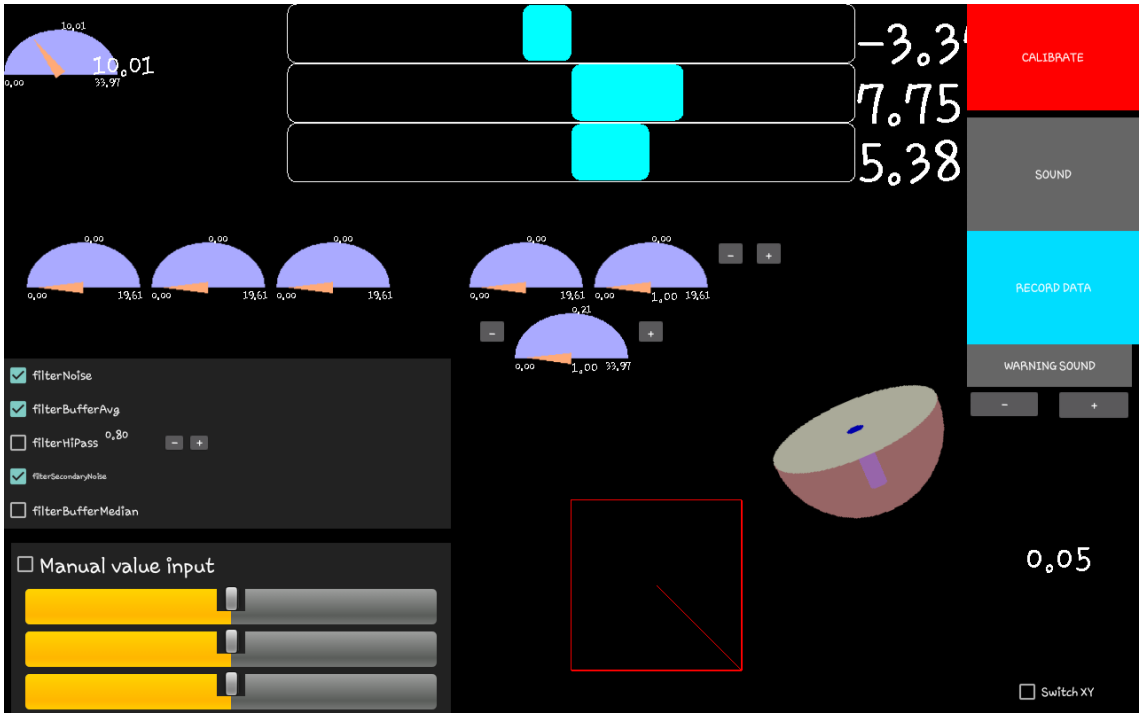
KUVA 8. Matkapuhelimen akselikääntö valinta

Jossain vaiheessa kehitystä ajattelin, että kaikkien asetusten ja säätöjen teko olisi kätevää sijoittaa omaan asetus valikkoonsa. Tästä ideasta luovuin kuitenkin melko nopeasti, sillä ohjelmaa on tarkoitus käyttää liikkuvassa ajoneuvossa ja tällöin valikoiden selaaminen saattaa olla paitsi laitonta myös vaarallista.

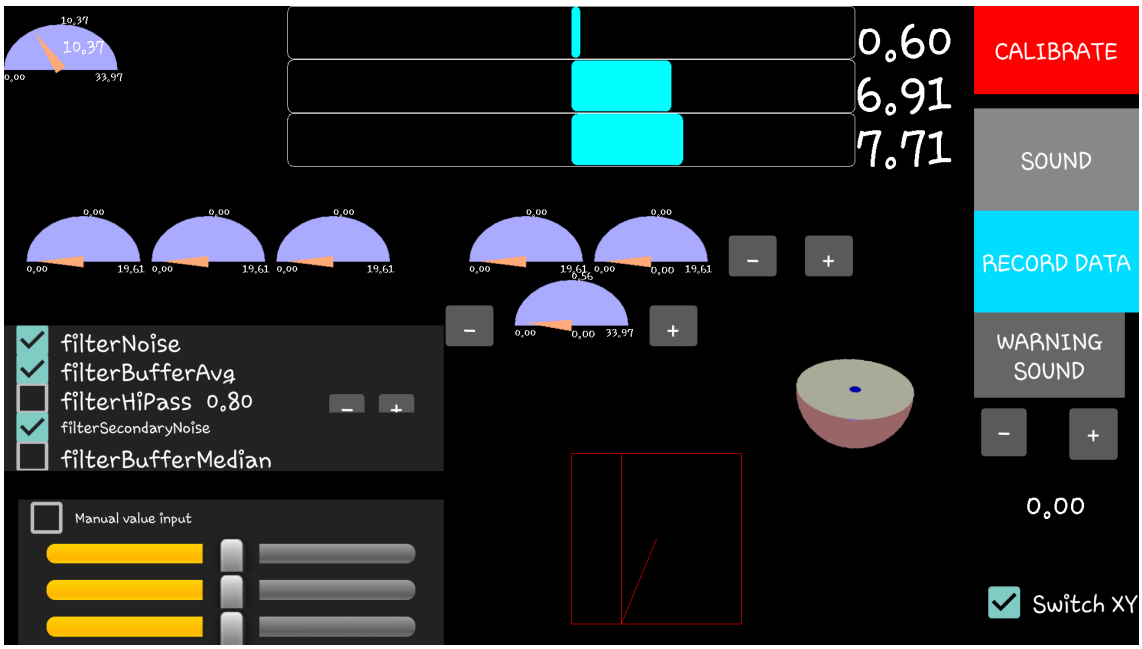
Android Studiossa on verrattain kätevä, sisäänrakennettu ulkoasueditori jolla on helppo ja nopea lisätä näytölle perus käyttöliittymäelementtejä, kuten painikkeita. Toisen testilaitteen saatuani kuitenkin havaitsin, että tämä editori kätevyydestään huolimatta aiheuttaa suurta mieliharmia. Elementtien kokoa ei voi nimittäin asettaa prosenttiarvoina, vaan yksikkönä toimii px, in, mm, pt, dp tai sp (6). Lyhyesti arvojen merkitys on seuraava:

- px – pikseliä, yksi px on yksi pikseli ruudulla
- in – tuumia,
- mm – millimetriä
- pt – pistettä, 1/72 tuumaa
- dp – näytön pikselitiheydestä riippumaton arvo. Lasketaan perustuen näytön dpi (dots per inch, pistettä tuumalla) arvoon.
- sp – kuten dp mutta käyttäjän asettama kirjasimen koko vaikuttaa.

Suositeltuna yksikkönä elementtien kokoa asetettaessa on dp. Ajatuksena on että painikkeet pysyvät käyttökelpoisen kokoisena näytön koosta riippumatta. Tämä kuitenkin aiheuttaa suuria ongelmia, jos yritetään mahduttaa ruudulle monta tällä tavoin määriteltyä elementtiä. Se mikä näyttää suurinäyttöisen tabletin näytöllä hyvältä, saattaa pieneltä matkapuhelimen ruudulta ollakin sekasotku kaikkien elementtien limittyessä toistensa päälle. Huomasin ongelman vasta niin myöhäisessä vaiheessa, että en ryhtynyt säätämään elementtejä ohjelmallisesti vaan pyrin jonkinlaiseen kompromissiin, jotta käyttöliittymä olisi käyttökelvollinen molemmissa testilaitteissani (kuvat 9 ja 10).



KUVA 9. Käyttöliittymä tabletin näytöllä



KUVA 10. Käyttöliittymä matkapuhelimen näytöllä.

4 TOTEUTUS

Tein työn kokonaan Linux-ympäristössä, sillä Android Studion emulaattorin ajaminen Windowsissa AMD-prosessorilla on todella hidasta. Pelkkä emuloidun järjestelmän käynnistäminen vei useita minuutteja, kun taas Linuxissa tämä tapahtui muutamassa sekunnissa.

4.1 Testaus

Käyttöliittymän komponenttien testaaminen onnistui emulaattorissa syöttämällä manuaalisesti arvoja, jotka simuloivat kiihtyvyysanturilta tulevaa dataa. Varsinainen toiminnallisuuden testaaminen tapahtui käytännössä ajelemalla autolla ja nauhoittamalla dataa laitteen muistiin myöhempää analysoimista varten. Testaukset suoritin iltahämärissä hiljaisilla tieosuuksilla ja alueilla, joissa ei ihmisiä juuri liiku. Jatkuva kaasuttelu ja jarruttelu olisi hyvin vahvasti saattanut häiritä muuta liikennettä.

4.2 Activity- ja view-luokka

Android-ympäristössä ohjelman pohjana on activity-luokka, joka on ikään kuin ruutunäkymän taustalla pyörivä osio johon sisällytetään toiminnallisuuteen liittyvä koodi. Sensoreiden hallinta, käyttäjän kosketukset, painikkeiden toiminnot sekä muut toiminnallisuudet sijoitetaan activityyn. Activity kannattaa ajatella yhtenä ruutunäkymänä. Yksin activity ei kuitenkaan ole kuin musta ruutu. Tarvitaan view-luokka. View on tavallaan työpöytäympäristöistä tuttu ikkuna, johon piirretään tarvittavat painikkeet ja graafiset elementit, jotka ruudulla tulee näkyä. View voi olla minkä kokoinen tahansa. Asetusten säätämistä varten voisi olla pieni view ja peleissä voi ponnahtaa ruudulle pieni view ilmaisemaan pelin päättymisestä tai kentän läpäisemisestä. Yleensä view on kuitenkin koko ruudun peittävä elementti, johon varsinaiset, ohjelman kannalta tarpeelliset grafiikat ja käyttöliittymäelementit sijoitetaan näkyville.

Tämä sovellus sisältää vain yhden activityn ja yhden viewin. Activity sisältää sensorin alustamiseen ja sen lukemiseen liittyvää koodia, kaikki suodattimet ja muun datan prosessointiin sekä näppäinten painalluksiin ja äänen prosessointiin liittyvät toimenpiteet. Viewissä luodaan ainoastaan käyttöliittymä objektit, joille annetaan arvot activityltä.

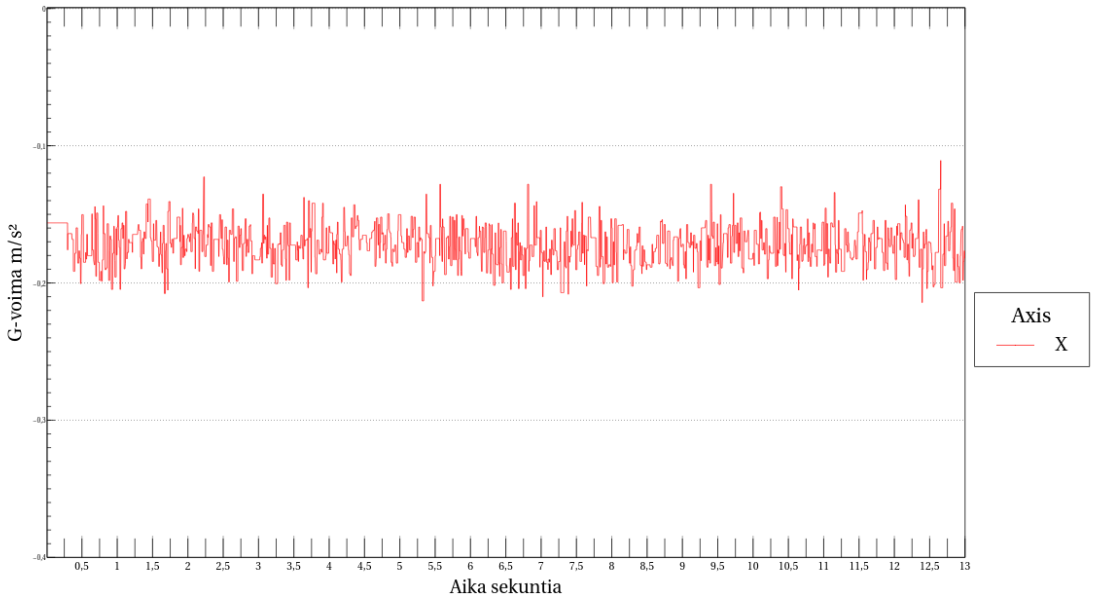
4.3 Kiihtyvyysanturi

Kiihtyvyysanturi otetaan käyttöön ja sieltä saadaan dataa seuraavasti:

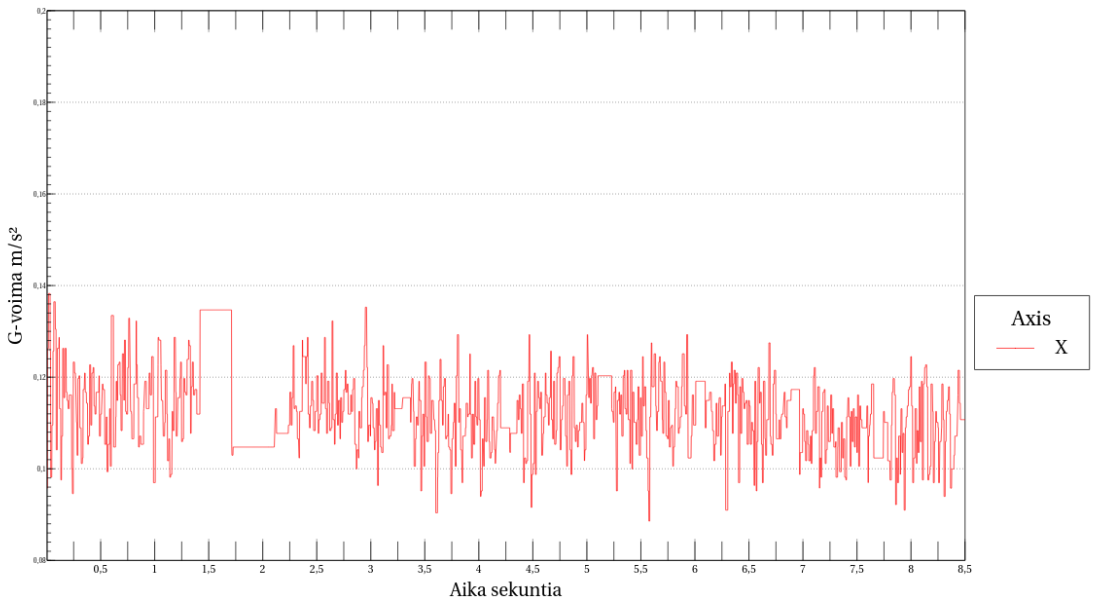
```
SensoriHomma = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
SensoriHomma.registerListener(this, AcceleroMylly, SensorManager.SENSOR_DELAY_FASTEST);



   @Override
    public void onSensorChanged(SensorEvent SensorEvent) {
        if (SensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
               float  sensroValueArray[] = SensorEvent.values;
       ...
       }
}
```

Joka kerta kun järjestelmä ilmoittaa sensorin datan muuttuneen, tarkistetaan, onko kyseessä kiihtyvyysanturi ja sijoitetaan uudet arvot sensorValueArray taulukkomuuttujaan.

Ilman suodattimia datassa ilmenee pientä häiriötä, jonka määrä riippuu mobiililaitteen kiihtyvyysanturin laadukkuudesta (kuvat 11 ja 12).



KUVA 11. Samsung Galaxy Tab 4 10.1:n kiihtyvyysanturin häiriö



KUVA 12. Samsung Galaxy S4:n kiihtyvyysanturin häiriö

Kuvissa on nähtävissä käytössäni olleen Samsung Galaxy S4 -puhelimen sekä Samsung Galaxy Tab 4 10.1 -tabletin tuottama häiriö yhdeltä akselilta laitteiden ollessa levossa pöydällä. Puhelimen datasta on myös nähtävissä, kuinka taus-

talla pyörivät sovellukset hidastavat puhelimen toimintaa ja aiheuttavat mittaukseen kanttiaaltona havaittavaa häiriötä.

4.4 Suodattimet

Erilaisia suodattimia kokeilemalla sain häiriön muutettua tasaiseksi viivaksi. Parhaimman lopputuloksen sain yhdistämällä kolme suodatinta.

4.4.1 Ensimmäinen kohina-arvosuodatin

Ensimmäinen suodatin toimii siten että vertaillaan kahta kiihtyvyysanturin arvoa. Jos uusi arvo ylittää edellisen arvon kiinteästi määritellyn kohina-arvon verran, sijoitetaan uusi arvo muuttujaan, jota käytetään seuraavan kerran vertailukohteena. Useiden testien jälkeen kohina-arvoksi valikoitui 0,05 m/s². Koodiesimerkissä ensin tarkistetaan suodattimelle kuuluvan valintaruudun tila, jonka jälkeen kiihtyvyysanturin taulukkomuuttuja käydään läpi ja jokaisen akselin kohdalla tarkistetaan, ylittääkö edellisen ja uuden arvon erotus kohina-arvon:

```
if (checkBoxFilter1.isChecked()){
   for (int i=0;i<=2;i++){
      if (Math.abs(accelPrev[i] - sensroValueArray[i]) > accelNoise) {
       accelPrev[i] = sensroValueArray[i];
      }
      else {
       sensroValueArray[i] = accelPrev[i];
      }
   }

}
```

4.4.2 Keskiarvosuodatin

Toinen suodatin puskuroi jatkuvasti viisi arvoa kiihtyvyysanturilta ja laskee näiden keskiarvon:

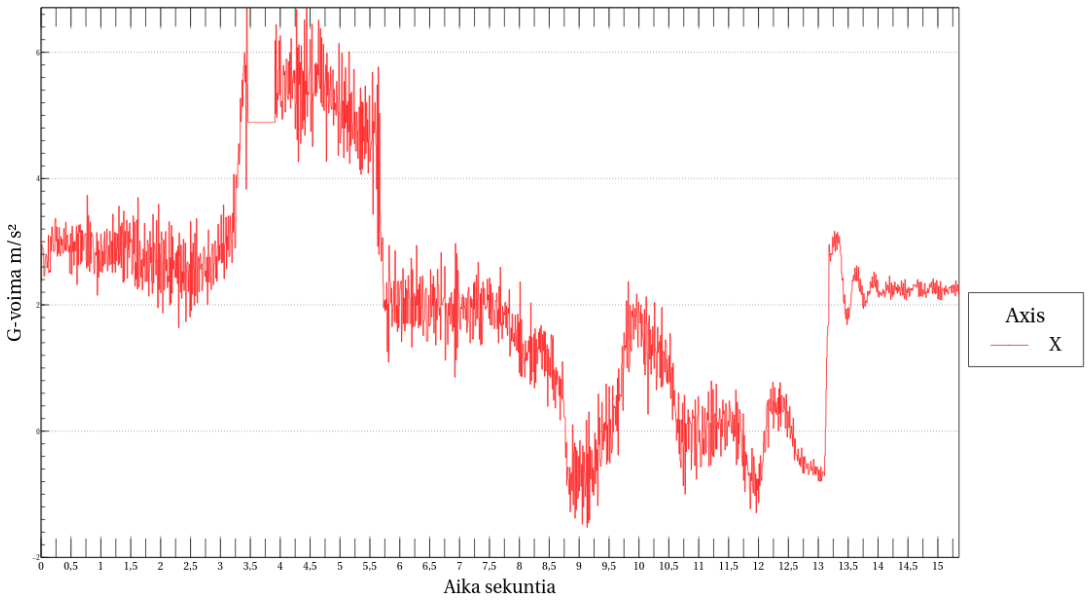
```
if (checkBoxFilter2.isChecked()) {
	float xTemp = 0;
	float yTemp = 0;
	float zTemp = 0;

	acceleratorBufferX.add(sensroValueArray[0]);
	acceleratorBufferY.add(sensroValueArray[1]);
	acceleratorBufferZ.add(sensroValueArray[2]);

	accelBufferRdy = (acceleratorBufferX.size() == accelBufferSize)
			? true : false;

	if (accelBufferRdy) {
		for (int i = 0; i < accelBufferSize; i++) {
		xTemp += acceleratorBufferX.get(i);
		yTemp += acceleratorBufferY.get(i);
		zTemp += acceleratorBufferZ.get(i);

	}

		sensroValueArray[0] = xTemp / accelBufferSize;
		sensroValueArray[1] = yTemp / accelBufferSize;
		sensroValueArray[2] = zTemp / accelBufferSize;

		acceleratorBufferX.remove(0);
		acceleratorBufferY.remove(0);
		acceleratorBufferZ.remove(0);

	}
}
```

4.4.3 Toinen kohina-arvosuodatin

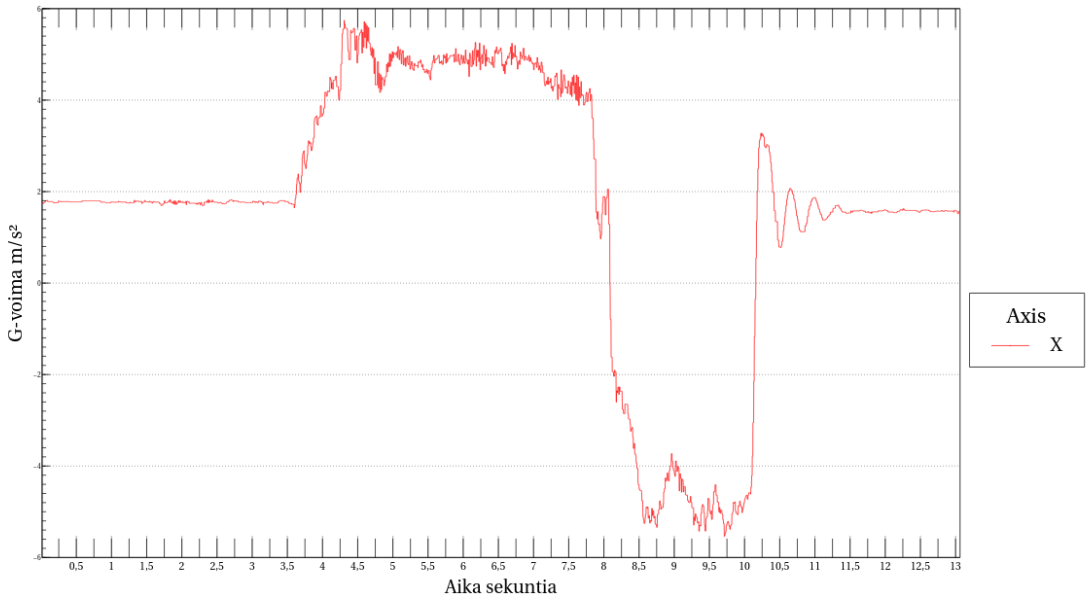
Kolmas suodatin toimii samoin kuin ensimmäinenkin mutta jo kahden suodattimen läpi kulkenut data sisältää niin vähän kohinaa, että kolmannelle suodattimelle kohina-arvoksi riittää 0,03 m/s². Kolmannen suodattimen tarkoitus onkin pääasiassa vain tasata keskiarvosuodatuksen dataa.

```
if (checkBoxFilter4.isChecked()){
   for (int i=0;i<=2;i++){
        if (Math.abs(accelPrev[i] - sensroValueArray[i]) > accelSecondaryNoise) {
        accelPrev[i] = sensroValueArray[i];
        }
        else {
        sensroValueArray[i] = accelPrev[i];
        }
   }

}
```

Näiden suodattimien käytöstä on selvä hyöty dataa analysoitaessa (kuvat 13 ja 14).



KUVA 13. Kiihdytys ilman suodattimia



KUVA 14. Kiihdytys suodattimien kera

Molemmissa kuvissa on miltei samankaltainen rivakka kiihdytys jonka jälkeen välittömästi jarrutus, joskin ilman suodattimia suoritetussa mittauksessa jarrutus ei ole yhtä voimakas, vaan se jakautuu hieman pidemmälle ajan jaksolle. Suodatetussa datassa kohina on huomattavasti pienempää, joten siitä on helpompi reaaliajassa havaita voimakkaan kiihdytyksen alkamisajankohta.

4.4.4 Mediaanisuodatin

Muita kokeellisia suodattimia oli mediaanisuodatin, jossa puskurin keskiarvon sijaan näytepuskurista otettiin keskiluku:

```
if (checkBoxFilter5.isChecked()) {

accBuffMedianX.add(sensroValueArray[0]);
accBuffMedianY.add(sensroValueArray[1]);
accBuffMedianZ.add(sensroValueArray[2]);

accelBufferMedianRdy = (accBuffMedianX.size() == accelBufferMedianSize) ? true : false;

       if (accelBufferMedianRdy) {
              float[] temparrayX = new float[accelBufferMedianSize];
```

```
            float[] temparrayY = new float[accelBufferMedianSize];
            float[] temparrayZ = new float[accelBufferMedianSize];

            for (int i = 0; i < accelBufferMedianSize; i++) {

            temparrayX[i] = accBuffMedianX.get(i);
            temparrayY[i] = accBuffMedianY.get(i);
            temparrayZ[i] = accBuffMedianZ.get(i);
            }

            Arrays.sort(temparrayX);
            Arrays.sort(temparrayY);
            Arrays.sort(temparrayZ);

            if (accelBufferMedianSize %2 == 0) {
            sensroValueArray[0] = (temparrayX[temparrayX.length/2] +
                                   temparrayX[temparrayX.length/2-1])/2;
            sensroValueArray[1] = (temparrayY[temparrayY.length/2] +
                                   temparrayX[temparrayY.length/2-1])/2;
            sensroValueArray[2] = (temparrayZ[temparrayZ.length/2] +
                                   temparrayX[temparrayZ.length/2-1])/2;
            } else {
            sensroValueArray[0] = temparrayX[temparrayX.length/2];
            sensroValueArray[1] = temparrayY[temparrayY.length/2];
            sensroValueArray[2] = temparrayZ[temparrayZ.length/2];
            }

            accBuffMedianX.remove(0);
            accBuffMedianY.remove(0);
            accBuffMedianZ.remove(0);
       }
}
```

Tämä suodatin vääristi todellisia lukemia niin paljon että totesin sen käyttökelvottomaksi.

4.4.5 Matalataajuussuodatin

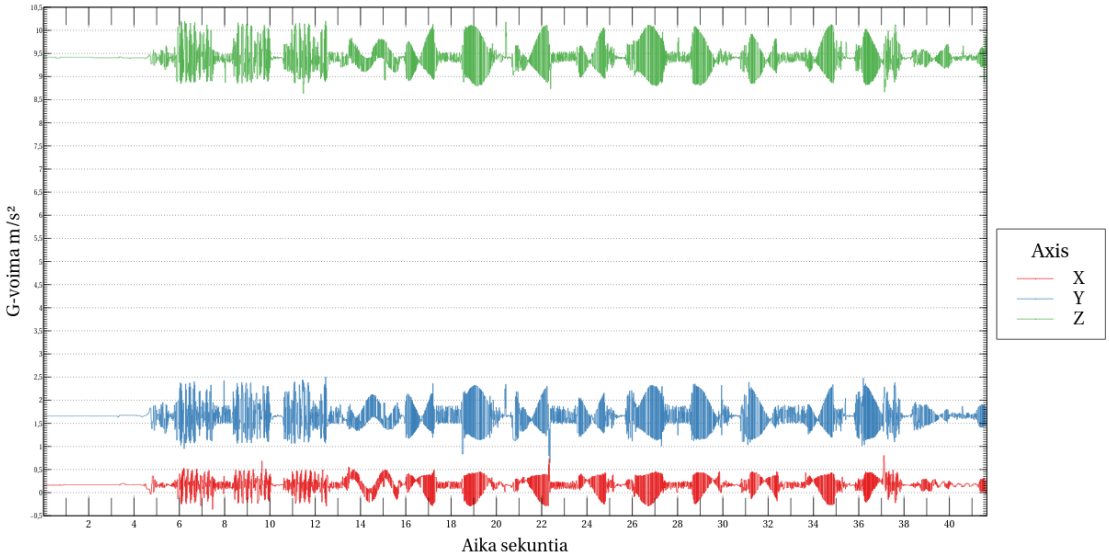
Toinen kokeellinen suodatin oli matalataajuussuodatin, eli korkeapäästösuodatin, joka tehokkaasti eliminoi painovoiman vaikutuksen anturin akseleilta.

```
if (checkBoxFilter3.isChecked()) {

        float alpha =(float) hiPassFilterAlphaValue/100;

        gravity[0] = alpha * gravity[0] + (1 - alpha) * sensroValueArray[0];
        gravity[1] = alpha * gravity[1] + (1 - alpha) * sensroValueArray[1];
        gravity[2] = alpha * gravity[2] + (1 - alpha) * sensroValueArray[2];

        sensroValueArray[0] = sensroValueArray[0] - gravity[0];
        sensroValueArray[1] = sensroValueArray[1] - gravity[1];
        sensroValueArray[2] = sensroValueArray[2] - gravity[2];
}
```

Suodattimen alpha-arvo on muutettavissa käyttöliittymästä (kuva 5). Tällainen data on mahdollista saada myös suoraan mobiililaitteen gyroskoopilta mutta toisessa testilaitteessani ei gyroskooppia ollut ja työn määrittelyn mukaan datan mittaus piti suorittaa nimenomaan kiihtyvyysanturia käyttäen. Suodatin ei myöskään tehnyt suurta vaikutusta sillä se neutraloi myös pitkään kestävän voimakkaan kiihdytyksen. Ainoastaan nopeat piikit G-voimissa pääsivät suodattimesta läpi ja tällainen data ei ollut kovin käyttökelpoista liiallisen kiihdytyksen mittaamisessa.

4.5 Häiriötekijöitä

Aiemmin esitelty kuva 14 on lähestulkoon ihanteellisesta mittaustilanteesta tasaisella asfaltilla. Normaaliajossa näin selkeää dataa on vaikea saada ja kiihdytysten erottaminen kuopista ja ylämäistä on huomattavasti hankalampaa. Myös irtosora tai pelkkä äänentoistolaitteiston liian suuri äänenvoimakkuus aiheuttaa häiriötä dataan (kuva 15).



KUVA 15. Äänentoistolaitteiston vaikutus

Kuvan mittaus on suoritettu pysäköidyssä autossa moottorin ollessa sammutettuna. Äänentoiston häiriö on huomattavasti suurempaa, kuin kiihtyvyysanturin sisäinen kohina.

4.6 Magnitudi

Liian voimakasta kiihtyvyyttä tarkkaillaan laskemalla kiihtyvyysanturin arvoista magnitudi kaavalla (1)

$$\sqrt{x^2+y^2+z^2} \quad (1),$$

missä

x on x-akseli,

y y-akseli,

z z-akseli

Laitteen ollessa levossa magnitudi ilmaisee laitteeseen kohdistuvan painovoiman. Androidin kirjastossa on painovoimalle oma vakiomuuttuja, jonka arvo on 9,80665 m/s². Kun kiihtyvyysanturin datan magnitudista vähennetään tämä pai-

novoimakomponentti, teoriassa jäljelle jää ajoneuvon kiihdytyksestä johtuva kiihtyvyys arvo. Tämä arvo on kuitenkin niin pieni, että siitä on todella vaikea erottaa, onko kyseessä todellinen kiihdytys tilanne vai tien epätasaisuudesta johtuva tärinä. Mittauksessa (Kuva 16) onkin nähtävissä, että varsinainen kiihdytys hädin tuskin erottuu mutta jarrutus jossa kiihtyvyys kasvaa reilummin, on rekisteröitynyt oikein. Tämä johtuu siitä että kiihtyvyysanturin mittaama painovoima ei vastaa Androidin kirjaston painovoimavakiota.

4.7 Kalibrointi

Kalibroimalla saadaan selville anturin lepo arvot, joita vertaamalla reaaliaikaiseen dataan, voidaan saada parempi käsitys kiihtyvyydestä.

Kalibrointitoiminto mittaa dataa joko 1000 näytettä tai 1000 millisekuntia, riippuen siitä kumpi tulee ensin täyteen. Koska anturia mitataan anturin ilmoittamalla maksimi näytteenottotaajuudella, nopeammalla anturilla 1000 näytettä voi hyvinkin tulla täyteen alle sekunnissa. Hitaammalla anturilla taas kalibrointi veisi kohtuuttomasti aikaa, jos joudutaan odottamaan useita sekunteja näytteenottopuskurin täyttymistä. Kun puskuri tai aikaraja on täynnä, lasketaan joka akselille keskiarvo, joka tallennetaan kyseisen akselin nollatasoksi. Näin saadaan joka akselille nollataso. Joka akselin mittausarvosta vähennetään tämä nollataso, muutetaan erotus positiiviseksi ja lasketaan yhteen kaavalla (2)

$$\sqrt{(x-x_0)^2}+\sqrt{(y-y_0)^2}+\sqrt{(z-z_0)^2} \quad (2),$$

missä

x on x-akseli,

x_0 x-akselin nollataso,

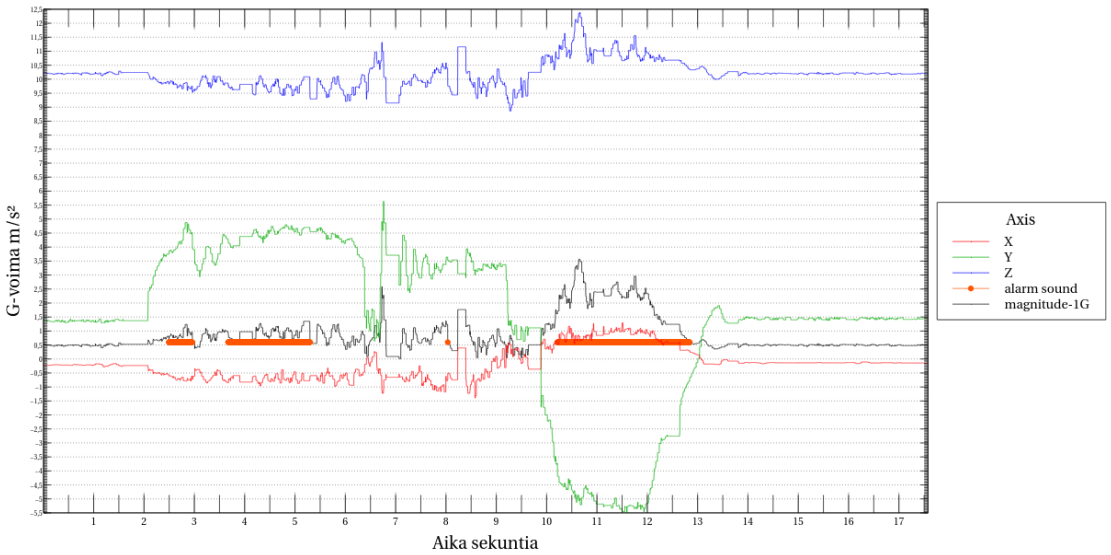
y y-akseli

y_0 y-akselin nollataso,

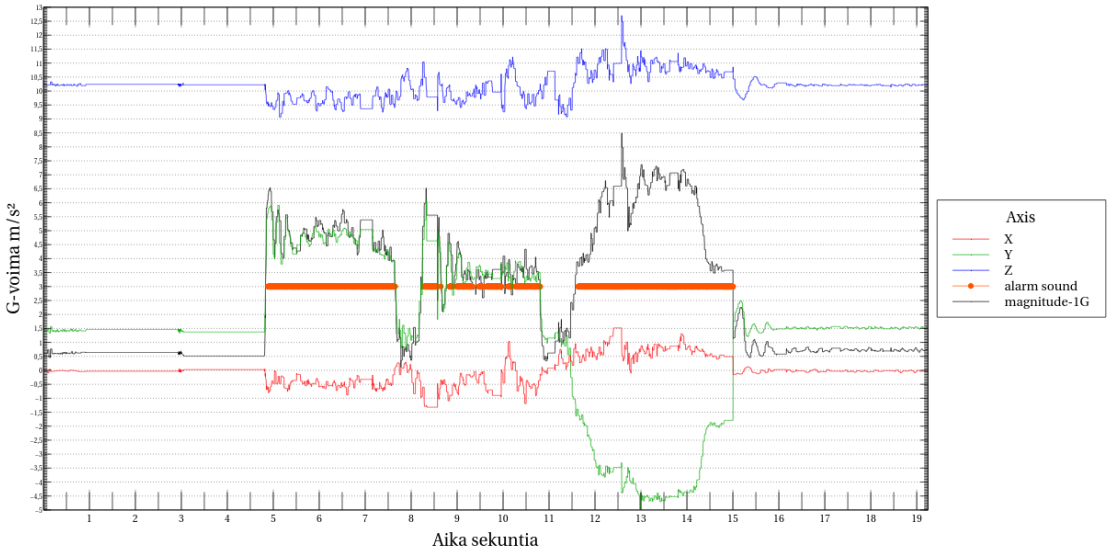
z z-akseli,

z_0 z-akselin nollataso

Näin saadaan selkeitä arvoja, jolloin todellisen kiihdytyksen erottaminen ylimääräisestä häiriöstä helpottuu huomattavasti. (kuva 17).

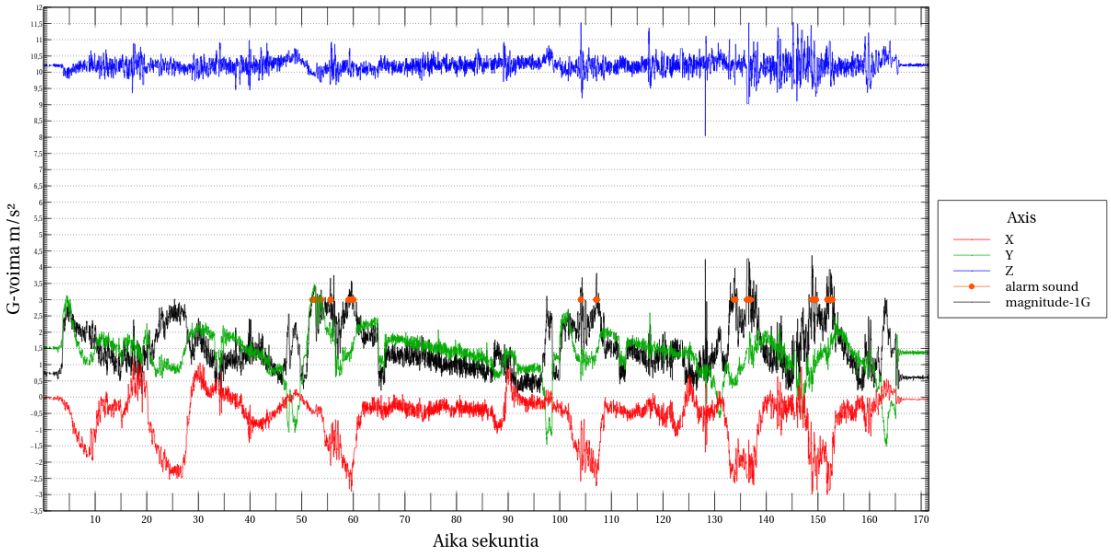


KUVA 16. Kiihdytys ilman kalibrointia



KUVA 17. Kiihdytys kalibroituna

Molemmissa kuvissa on samankaltainen kiihdytys tilanne. Liikkumaton ajoneuvo, rivakka kiihdytys ykkösvaihteella, vaihteen vaihto kakkoselle ja voimakas jarrutus niin että ajoneuvo on täysin pysähdyksissä. Mustalla piirretty viiva kertoo aikaisemmin mainituilla kaavoilla lasketun mitta-arvon, jonka perusteella päätetään, onko kyseessä kiihdytys. Oranssi paksumpi viiva kertoo, milloin kyseessä on liian voimakas kiihdytys ja varoitusäänimerkki soi. Kalibroimattoman mittauksen arvoissa liiallinen kiihdytys tunnistettiin, kun kiihtyvyysanturin arvot olivat 0,3 m/s² korkeammalla kuin magnitudi josta painovoima oli vähennetty, vähintään 0,5 sekuntia kestävän ajanjakson. Kalibroidussa mittauksessa vastaavat arvot olivat 3 m/s² ja 50 ms. Molempia arvoja voisi hieman nostaa jolloin varoitus ääntä ei annettaisi niin herkästi esimerkiksi kuoppaan ajettaessa (kuva 18). Riittävän jyrkässä ylämäessä varoitus ääni kuitenkin saattaisi kuulua.

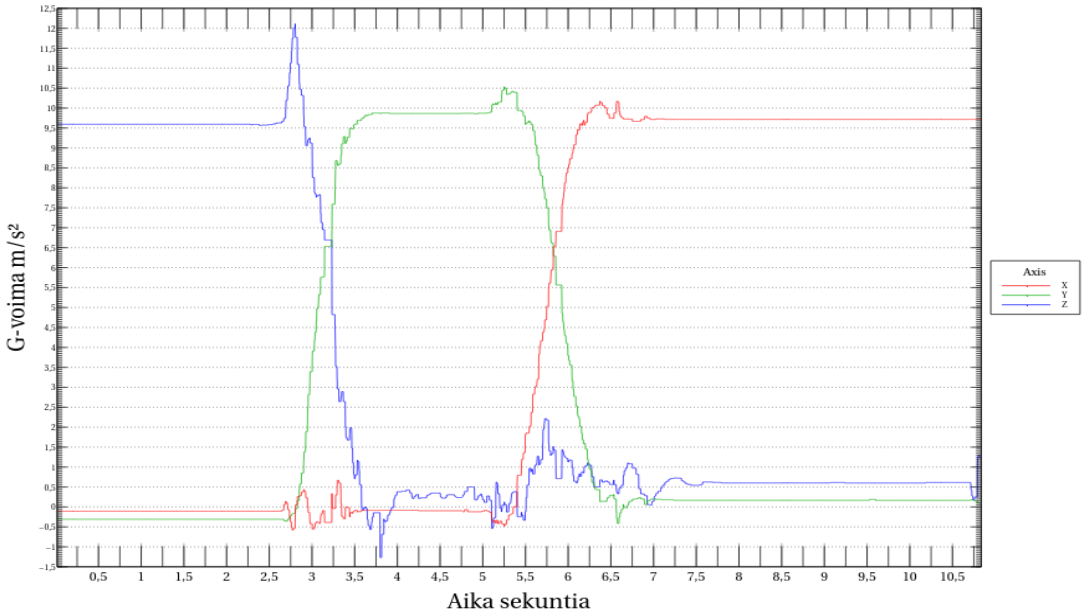


KUVA 18. Lyhyt rauhallinen ajo.

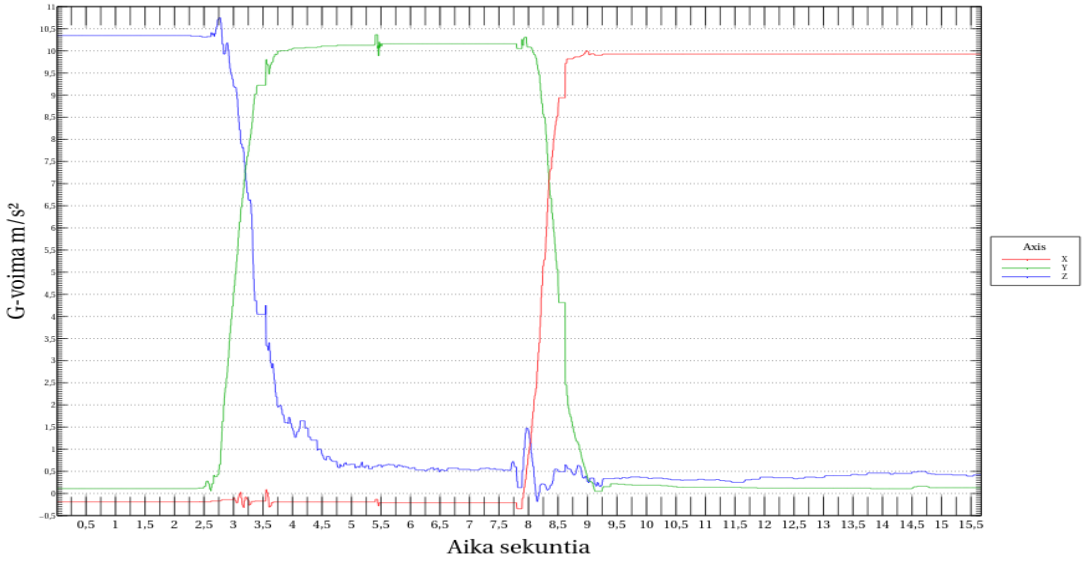
Kuvassa on dataa lyhyestä ajosta, jossa kiihdytykset ja jarrutukset olivat hyvin rauhallisia. Kuoppiin ajettaessa varoitusääni kuului.

5 TULOSTEN ANALYSOINTIA

Androidin kirjastosta löytyy painovoima vakioita eri planeetoille. Myös fiktiivinen ”death star 1” planeetta on varmuudenvuoksi kirjastoihin lisätty. Sen painovoimavakioksi ilmoitetaan 3,5303614E-7 imperiumin yksikköä (m/s²). Maapallon painovoimavakioksi on määritelty 9,80665 m/s². (7.) Tämä ei nähtävästi sovi yhteen mobiililaitteiden sensoreiden antamien lukemien kanssa. Joka laitteessa kiihtyvyysanturi antaa hieman eri lukeman painovoimaksi. Jopa laitteen akseleiden välillä on eroa mittauksen suhteen (kuvat 19 ja 20).



KUVA 19. Samsung Galaxy Tab 4 10.1:n kiihtyvyysanturin lepoarvot



KUVA 20. Samsung Galaxy S4:n kiihtyvyysanturin lepoarvot.

Molemmissa kuvissa laite on lepoasennossa pöydällä josta nostin sen pitempi sivu pöytää vasten josta taas kapea sivu pöytää vasten pystyasentoon. Kuvassa 19 tablettilaiteen kiihtyvyysanturin mittaama painovoima vaihtelee noin 9,55–9,86 m/s². Alemmassa kuvassa taas puhelimen arvot vaihtelevat välillä 10,34–9,92 m/s² mikä on joka akselilta hieman liikaa verrattuna Androidin kirjaston painovoimavakioon.

Kiihtyvyysanturin toimintaa voi pohtia ajattelemalla kiihtyvyys anturin toimivan samalla tavalla kuin ajoneuvon kattoon sisäpuolelle kiinnitetty venyvä naru jonka päässä on pallo. Tarkkailemalla pallon liikkeitä, voidaan päätellä, milloin kiihtyvyyttä tapahtuu, mutta myös kuoppa tiessä aiheuttaa pallon heilumista. Ylämäessä pallon liike näyttää samalta kuin kovassa kiihdytyksessä. Tällaisen pallon heilumista tarkkailemalla voidaan kovat kiihdytykset päätellä jotakuinkin yhtä hyvin kuin matkapuhelimen halvan kiihtyvyysanturin datan perusteella.

Kovan kiihdytyksen määrittäminen ylipäätään on hieman kyseenalaista. Vanhalla tehottomalla autolla kiihdytys ei ole kova, vaikka kaasun painaisi pohjaan. Polttoainetta kuluu silti todella paljon. Uudella turboahdetulla autolla taas rauhallinen kiihdytys saattaa vastata vanhemman auton täyttä kiihtyvyyttä, mutta se kuluttaa huomattavasti vähemmän polttoainetta.

Pelkkää kiihtyvyysanturia käyttäen on mahdotonta kertoa mitään ajoneuvon teknisistä ominaisuuksista johtuvasta polttoaineen kulutuksesta, mutta kuljettajan ajokäyttäytymistä epätaloudellisen ajamisen alueella voidaan kuitenkin joltain osin tutkia.

6 POHDINTA

Työn alkuperäinen määritelmä oli tuottaa Android-sovellus joka mittaa kiihtyvyysanturilla polttoainetaloutta ja kuljettajan ajokäyttäytymistä. Tähän tavoitteeseen pääsin osittain. Sovelluksella voi havaita voimakkaat kiihtyvyyden muutokset mutta niiden tarkkoja arvoja on mahdoton saada. Tarkka mittaus olisi polttoainetalouden määrittämiseksi perusedellytys.

Keskityin alussa liikaa erilaisten suodattimien tekemiseen, sillä oletin anturin häiriön olevan niin suurta, että se vaikuttaisi huomattavasti mittaustuloksiin. Todellisuudessa anturin sisäinen häiriö on kuitenkin niin pientä, että todellisessa kiihdytys tilanteessa sitä ei juurikaan havaitse, koska auton kiihtyvyyden tuottamat G-voimat ovat kymmeniä kertoja suurempia kuin häiriö itse. Suurin häiriö syntyy auton alustasta, moottorin tärinästä ja tien epätasaisuudesta. Yksinkertaiset suodattimet joilla lasketaan kohinan keskiarvo, osoittautuivat parhaimmiksi tavoiksi tasata mittaus dataa. Tämän lisäksi käytin liiaksi aikaa pohtiessani ja testaillessani erilaisia tapoja joilla olisi saatu laskettua jonkinlainen suuntaa antava polttoaineen kulutus. Tämä saattaisi olla teoriassa mahdollistakin mutta vaatisi suunnattomasti lisää suunnittelua ja testausta. Pelkkään kiihtyvyysanturiin perustuen tuloksista tulisi joka tapauksessa äärimmäisen epätarkkoja. GPS-datan liittäminen laskelmiin olisi huomattavasti parempi tapa lähestyä tällaisen toiminnallisuuden aikaan saamista.

Kiihtyvyysanturin nauhoitettua dataa tarkastelemalla jälkeenpäin voidaan saada jonkinlainen käsitys ajon aikaisista tapahtumista. Datasta on mahdotonta seuloa pois ylimääräiset häiriöt, esimerkiksi soralla ajon aiheuttama tärinä tai kuoppaan ajettaessa aiheutuva voimakas piikki G-voimissa. Joten reaaliaikainen, tarkka, datan analysointi aivan liian työläs toteuttaa.

Riittävän hyvällä kiinnityksellä matkapuhelinta ja tämän työn tuloksena syntynyttä sovellusta voi kuitenkin käyttää muistuttamaan milloin on aika hieman keventää kaasujalkaa. Varoitusäänimerkkiin liittyvien arvojen asettaminen jää kuiten-

kin käyttäjän vastuulle, vaikkakaan niitä säätämällä ei todennäköisesti miten-kään voida eliminoida virheellisiä varoitusääniä. Sovellusta en tällaisenaan lataa mihinkään palveluun kenenkään saataville sillä sovellus ei käytettävyydeltään ole mitenkään erinomainen.

Käyttöliittymän parantamiseen voisi kuluttaa vaikka kuinka paljon aikaa mutta se olisi jo täysin oma työkokonaisuutensa. Ohjelman toimintojakin pitäisi lisätä huomattavasti, jotta sovelluksesta olisi varsinaista iloa. Sovellus voisi esimerkik-si antaa ajon jälkeen raportin keräämästään datasta ja dataa olisi miellyttäväm-pää analysoida suoraan sovelluksesta. Toisaalta taas minkäänlaisen mobiililait-teen silmäily ajotilanteessa ei ole kenenkään eduksi, joten en voi suositella täl-laista sovellusta kenellekään.

LÄHTEET

1. OpenSUSE. 2018. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/OpenSUSE. Hakupäivä 5.4.2018
2. Main Page. 2018. OpenSUSE. Saatavissa: https://en.opensuse.org/Main_Page. Hakupäivä 5.4.2018
3. Meet Android Studio. 2018. Android Developers. Saatavissa: https://developer.android.com/studio/intro/. Hakupäivä 5.4.2018
4. Introduction. 2018. Veusz 2.0 documentation. Saatavissa: https://veusz.github.io/docs/manual/introduction.html. Hakupäivä 5.4.2018
5. Save files on device storage. 2018. Android Developers. Saatavissa: https://developer.android.com/training/data-storage/files. Hakupäivä 5.4.2018
6. More resource types. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/topics/resources/more-resources#Dimension. Hakupäivä 5.4.2018
7. SensorManager. 2018. Android Developers. Saatavissa: https://developer.android.com/reference/android/hardware/SensorManager.html. Hakupäivä 5.4.2018